MAC323 EXERCÍCIO-PROGRAMA 2

LIMIAR DE CONEXIDADE PARA CERTOS GRAFOS GEOMÉTRICOS II

Y. KOHAYAKAWA

Data de entrega: 29/4/2013 (23:55) [Adiado para 6/5/2013 (23:55)]

1. Introdução

Este EP é uma continuação do EP1. Naquele EP, você trabalhou no quadrado unitário $[0,1]^2 = [0,1] \times [0,1] \subset \mathbb{R}^2$; aqui, você trabalhará em outros espaços geométricos. Fixe um inteiro D positivo; vamos considerar $\mathbb{R}^D$ com a norma euclidiana $\|x\| = \langle x, x\rangle^{1/2}$, onde $\langle x, y\rangle$ é o produto interno usual: se $x = (x_i)_{1\le i\le D}$ e $y = (y_i)_{1\le i\le D}$, então $\langle x, y\rangle = x^T y = \sum_{1\le i\le D} x_i y_i$. Estamos particularmente interessados em valores grandes de D (isto é, espaços de *dimensão alta*). Consideraremos conjuntos de pontos contidos em dois subconjuntos particulares de $\mathbb{R}^D$, a saber, no *cubo*

$$C = [0, 1/\sqrt{D}]^D \subset \mathbb{R}^D \tag{1}$$

e na *esfera*

$$S = \{x = (x_i)_{1\le i\le D} \colon \|x\| = 1/2\} \subset \mathbb{R}^D. \tag{2}$$

Note que ambos C e S têm diâmetro 1 (o *diâmetro* de $X \subset \mathbb{R}^D$ é $\sup\{\|x-y\| \colon x,\, y \in X\}$).

2. Descrição do problema

No que segue, seja $X \subset \mathbb{R}^D$. Sejam dados N e $0 \le d \le 1$ e suponha $p_1, \dots, p_N \in X$. Definimos um grafo $G = (V, E)$ com $V = [N] = \{1, \dots, N\}$ e

$$E = \big\{\{i, j\} \colon \|p_i - p_j\| \le d\big\}. \tag{3}$$

Para explicitar a dependência de G da coleção de pontos p_i ($1 \le i \le N$) e de d, podemos escrever $G(\mathbf{p}, d)$ para G acima, onde $\mathbf{p} = (p_1, \dots, p_N)$. O *limiar de conexidade* $d_*(\mathbf{p})$ de $\mathbf{p}$ é o menor d tal que $G(\mathbf{p}, d)$ é conexo.

Seu EP deve implementar um algoritmo que, dado $\mathbf{p}$ como acima, determina $d_*(\mathbf{p})$. Este algoritmo deve ser 'eficiente' para instâncias aleatórias de $\mathbf{p}$ (como definido mais precisamente a seguir) com N e D 'grandes'.

2.1. Instâncias aleatórias.

Dado N, para gerar uma instância aleatória $\mathbf{p} = (p_1, \dots, p_N)$, você deve gerar os p_i ($1 \le i \le N$) uniformemente ao acaso em X, independentemente.

2.2. Limiar de conexidade típico.

Definimos (informalmente) o limiar de conexidade *típico* para N pontos aleatórios em X como sendo o valor 'típico' de $d_*(\mathbf{p})$, onde $\mathbf{p}$ é como definido em §2.1. Prova-se que, se $X = C$ ou $X = S$, quando N, $D \to \infty$ mas, digamos, $\log N = o(D)$, esses valores típicos convergem para um valor d_* ($0 < d_* < \infty$).

Versão de 24 de abril de 2013, 05:36.

3. Seu programa

Seu programa deve ter dois modos de operação.

3.1. Verificação de conexidade. No modo de *verificação de conexidade*, a tarefa principal do programa é verificar se um dado $\mathbf{p} = (p_1, \dots, p_N)$ e d definem um grafo $G(\mathbf{p}, d)$ conexo. Neste modo, seu programa deve receber como entrada D, N e d e ele deve gerar N pontos aleatórios como em §2.1. A saída do programa deve ser uma mensagem, dizendo se $G(\mathbf{p}, d)$ é conexo ou não. Seu programa deve também receber, opcionalmente, um inteiro não-negativo s (uma *semente*) para inicializar o gerador de números aleatórios (você deve usar `rand()` e `srand()` do `stdlib` em seu programa). Caso o usuário não dê uma semente, seu programa deve usar um valor padrão fixo de semente (um valor *default*). Finalmente, o usuário também deverá poder executar seu programa de modo que ele tenha também como saída uma listagem dos pontos p_i gerados (isto será útil na depuração de seu programa).

Seu programa deve receber a entrada na linha de comando. Assim, seu programa poderia ser executado, por exemplo, das seguintes formas:

```
prompt$ ep2 -D1000 -N100 -d0.5
prompt$ ep2 -D2000 -N1000 -d0.5 -s323
prompt$ ep2 -D5 -N10 -d0.5 -s323 -v
```

Você deve também implementar a seguinte forma de execução útil para depuração: com a chamada

```
prompt$ ep2 -D5 -C -d0.3
```

seu programa deve ler as coordenadas de pontos em $X \subset \mathbb{R}^D$ do `stdin` (no exemplo acima, $D = 5$ coordenadas devem ser lidas para cada ponto), e deve determinar se o d dado na linha de comando define um grafo conexo.

3.2. Estimação do limiar de conexidade típico. Seu programa também deve ser capaz de estimar o limiar de conexidade típico d_* (veja §2.2). Para tanto, o usuário fornecerá N e também um inteiro M. Seu programa deve gerar M instâncias aleatórias $\mathbf{p}_1, \dots, \mathbf{p}_M$ e deve determinar $d_*(\mathbf{p}_j)$ para todo $1 \leq j \leq M$. Seu programa deve então devolver como sua estimativa para d_* a média dos $d_*(\mathbf{p}_j)$ $(1 \leq j \leq M)$.

Novamente, o usário deve ter a opção de fornecer uma semente para o gerador de números aleatórios, através da opção de linha de comando `-s`. Ademais, se o usuário der a opção `-v`, seu programa deve imprimir os M valores $d_*(\mathbf{p}_j)$ $(1 \leq j \leq M)$. Ademais, implemente a opção `-V`, que faz com que seu programa imprima não só estes M valores, mas lista os $\mathbf{p}_j$ correspondentes.

Seu programa poderia ser executado, por exemplo, das seguintes formas:

```
prompt$ ep2 -D100 -N100 -M1
prompt$ ep2 -D3000 -N3000 -M10 -s323
prompt$ ep2 -D2000 -N2000 -M10 -s323 -v
prompt$ ep2 -D5 -N10 -M5 -s323 -V
```

Como em §3.1, você deve também implementar a seguinte forma de execução útil para depuração: com a chamada

```
prompt$ ep2 -D5 -L
```

seu programa deve ler uma seqüência de pontos em $X \subset \mathbb{R}^D$ do `stdin` e deve determinar $d_*(\mathbf{p})$, onde $\mathbf{p}$ é a seqüência de pontos dada pelo usuário.

3.3. **Cubo ou esfera; modularização.** Note que, em §§3.1 e 3.2, não especificamos se estamos tratando o caso $X = C$ ou $X = S$. Você deve escrever um programa, digamos `ep2.c`, que trata ambos os casos da mesma forma. Mais precisamente, você deve escrever um módulo `Point`, com a interface `Point.h` como abaixo:

```
/* Point.h */
typedef float *point;
point randPoint();
float distance(point, point);
...
```

Em `ep2.c`, você deve supor que `randPoint()` devolve um ponto gerado uniformemente ao acaso em X. Em `Point.C.c`, você deve implementar tal função `randPoint()` para $X = C$ e, em `Point.S.c`, você deve implementar tal função para $X = S$. Para tratar o caso $X = C$, você deve executar a ligação

```
prompt$ gcc -Wall -pedantic -ansi -o ep2 Point.C.o ep2.o -lm
```

e, para tratar o caso $X = S$, você deve executar a ligação

```
prompt$ gcc -Wall -pedantic -ansi -o ep2 Point.S.o ep2.o -lm
```

Note que `ep2.o` deve ser o *mesmo* arquivo nas duas ligações acima.

Caso você queira, seu programa pode ser decomposto em mais módulos (as ligações acima devem ser, naturalmente, adaptadas, mas a única diferença entre os casos $X = C$ e $X = S$ deve ser a ligação com `Point.C.o`/`Point.S.o`). Você também pode adicionar outras funções ao módulo `Point` (e portanto ter um arquivo `Point.h` maior). É natural representar pontos como vetores de D `floats`—entretanto, os módulos diferentes de `Point` não devem usar esta informação (isto é, pontos devem ser manipulados apenas através das funções disponibilizadas em `Point.h` (você deve tratar `point` como um tipo abstrato)).

3.4. **Observação sobre a geração de instâncias aleatórias.** É claro como gerar instâncias aleatórias como especificado em §2.1 no caso em que $X = C$. Estude o caso $X = S$ com cuidado. Depois de pensar nesse problema, você poderia discutir sua solução com seu professor de MAE228.

3.5. **Observação sobre eficiência.** Foi possível explorar a geometria do quadrado unitário no EP1 (por exemplo, para um dado d, era suficiente examinar $O(d^{-2}N^2)$ pares de pontos, em vez de examinar todos os $\binom{N}{2}$ pares (veja o programa 3.20 em §3.7 de nosso livro texto). Não é claro como usar essa idéia neste EP. Outra idéia importante para o EP1 foi usar boas estruturas de dados para representar partições de conjuntos (veja §4.5 de nosso livro texto).

Neste EP, estamos interessados em valores para D e N como, por exemplo, $(D, N) = (100, 100)$, $(100, 200)$, $(1000, 1000)$, $(1000, 3000)$, $(2000, 2000)$ (ou um pouco maiores).

Observações

1. *Este EP é estritamente individual.* Programas semelhantes receberão nota 0.
2. Seja cuidadoso com sua programação (correção, documentação, apresentação, clareza do código, etc), dando especial atenção a suas estruturas de dados. A correção será feita levando isso em conta.
3. Comparem entre vocês o desempenho de seus programas.
4. Entregue seu EP no Paca.
5. Não deixe de incluir em seu código um *relatório* para discutir seu EP: discuta as estruturas de dados usadas, os algoritmos usados, etc. *Se você escrever claramente como funciona seu EP, o monitor terá pouca dificuldade em corrigi-lo, e assim você terá uma nota mais alta.* (Se o monitor sofrer para entender seu código, sua nota será baixa.)

Observação final. Enviem dúvidas para a lista de discussão da disciplina.

Instituto de Matemática e Estatística, Universidade de São Paulo, Rua do Matão 1010, 05508–090 São Paulo, SP

Endereço eletrônico: `yoshi@ime.usp.br`